Proceedings of the Third International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2016)
Sofia, Bulgaria

Jesus Carretero, Javier Garcia Blas, Svetozar Margenov
(Editors)

October, 6-7, 2016

Marozzo, F., Duro, F. R., Garcia Blas, J., Carretero, J., Talia, D. & Trunfio, P. (2016). A Data-Aware Scheduling Strategy for DMCF workflows over Hercules. En *Proceedings of the Third International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2016) Sofia, Bulgaria* (pp. 37-44). Madrid: Universidad Carlos III de Madrid. Computer Architecture, Communications, and Systems Group (ARCOS).

A Data-Aware Scheduling Strategy for DMCF workflows over Hercules

Fabrizio Marozzo∗, Francisco Rodrigo Duro†, Javier Garcia Blas†

fmarozzo@dimes.unical.it, frodrigo@arcos.inf.uc3m.es, fjblas@arcos.inf.uc3m.es

Jesus Carretero†, Domenico Talia∗, Paolo Trunfio∗

jesus.carretero@uc3m.es, talia@dimes.unical.it, trunfio@dimes.unical.it

∗ DIMES, University of Calabria, Italy
† ARCOS, University Carlos III, Spain

Abstract

As data-intensive scientific prevalence arises, there is a necessity of simplifying the development, deployment, and execution of complex data analysis applications. The Data Mining Cloud Framework is a service-oriented system for allowing users to design and execute data analysis applications, defined as workflows, on cloud platforms, relying on cloud-provided storage services for I/O operations. Hercules is an in-memory I/O solution that can be deployed as an alternative to cloud storage services, providing additional performance and flexibility features. This work extends the DMCF-Hercules cooperation by applying novel data placement and task scheduling techniques for exposing and exploiting data locality in data-intensive workflows.

Keywords DMCF, Hercules, workflows, in-memory storage, data cache, Microsoft Azure, data locality

I. Introduction

Scientific computing applications and platforms are evolving from CPU-intensive tasks executed over strongly coupled infrastructures, i.e. complex simulations running on supercomputers, to data-intensive problems requiring flexible computing resources depending on the requirements and budget of the user. This evolution paves the future of Ultrascale systems, which will blur the differences of existing scientific computing infrastructures, such as HPC systems and cloud computing platforms. In current approaches, the interfaces and management of the different infrastructures are too different, requiring different programming models, even for the same application. In contrast, the future Ultrascale systems should take advantage of every possible resource available, in a transparent way for the user.

Workflow engines are the leading approach for executing data-intensive applications in different computing infrastructures. Scientific workflows consist of interdependent tasks, connected in a DAG style, which communicate through intermediate storage abstractions, typically files. There is a main tradeoff that should be taken into account when the user relies on workflow engines for data-intensive applications. While portability and flexibility offers a broader support of the existing computing resources, the achieved performance is usually limited in contrast with native applications (classical HPC applications running on HPC clusters or supercomputers).

The increasing availability of data generated by high-fidelity simulations and high-resolution scientific instruments in domains as diverse as climate, experimental physics, bioinformatics, and astronomy, has shown the underlying I/O subsystem to be a substantial performance bottleneck. While typical high-performance computing (HPC) systems rely on monolithic parallel file systems, data-intensive workflow implementations must borrow techniques from the Big Data computing (BDC) space, such as exposing data storage locations and scheduling work to reduce data movement. This lack of performance is the result of a sub-optimal exploitation of the available resources, based on two main reasons: task schedulers unable to select the best nodes depending on the characteristics of the task and under-performing I/O solutions.

Our previous works have targeted these disadvantages in a real-world scenario by combining two existing solutions: the Data Mining Cloud Framework (DMCF) and the in-memory

I/O accelerator known as Hercules. The present work deepens in this combination providing locality-aware features, both in the DMCF task scheduler and in the Hercules data placement algorithms. By running the workflow workers in the same VM instances as Hercules I/O nodes, data locality can be exposed and exploited, executing the task in the node where the data are stored in-memory.

This paper proposes the application of locality-aware data placement and data discovery techniques into the DMCF-Hercules integration. Additionally, this work proposes a novel task scheduler integrated in DMCF for the co-location of tasks and data, relying on the locality-aware functionality offered by Hercules. The evaluation carried on shows how data-locality exploitation is especially critical in cloud platforms, where virtualized network interfaces provide limited bandwidth in contrast with the state-of-the-art high-performance network infrastructures present in HPC systems.

The remainder of the paper is structured as follows. Section II describes the main features of DMCF. Section III introduces Hercules architecture and capabilities. Section IV emphasizes the advantages of integrating DMCF and Hercules and outlines how this integration will work. Section IV.3 details the novel locality-aware techniques proposed in this work. Section V presents preliminary results of the performance improvements achieved by the application of the locality-aware techniques in a Microsoft Azure cloud infrastructure. Finally, section VI concludes the work and give some future research related to the presented work.

II. Data Mining Cloud Framework overview

The Data Mining Cloud Framework (DMCF) [1] is a software system designed for designing and executing data analysis workflows on Clouds. A Web-based user interface allows users to compose their applications and to submit them for execution to the Cloud platform, following a Software-as-a-Service (SaaS) approach.

The architecture of DMCF includes different components that can be grouped into storage and compute components (see Figure 2).

The DMCF architecture has been designed to be implemented on top of different Cloud systems. The implementation used in this work is based on Microsoft Azure[1].

DMCF allows to program data analysis workflows using two languages: VL4Cloud (Visual Language for Cloud) and JS4Cloud (JavaScript for Cloud).

Both languages use two key programming abstractions:

- Data elements, denoting input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
- Tool elements, denoting algorithms, software tools or complex applications performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

Another common element is the Task concept, which represents the unit of parallelism in our model. A task is a Tool invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a data-driven task parallelism.

III. Hercules overview

Hercules [2] is a distributed in-memory storage system based on the key/value Memcached database [3]. The distributed memory space can be used by the applications as a virtual storage device for I/O operations and has been especially adapted in this work for being used as an in-memory shared storage for cloud infrastructures. Our solution relies on an improved version of Memcached servers, for offering an alternative storage solution to the default cloud storage service provided by Azure.

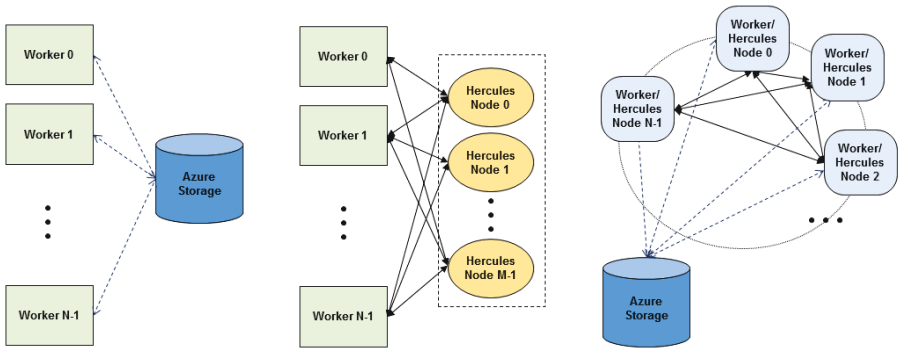
Figure 3 shows how Hercules architecture has two main layers: front-end (Hercules client library) and back-end (server layer). The worker user-level library is based on a layered design, while back-end components are based on the Memcached server, extending its functionality with persistence and tweaks. Main advantages offered by Hercules are: scalability, easy deployment, flexibility, and performance.

Scalability is achieved by fully distributing data and metadata information among all the nodes, avoiding the bottlenecks produced by centralized metadata servers. Data and metadata placement is completely calculated in the worker-side by a hash algorithm. The servers, on the other hand, are completely stateless.

Easy deployment and flexibility at worker-side are tackled using a POSIX-like user-level interface (open, read, write, close, etc.) in addition to classic put/get approach existing in current NoSQL databases. Existing software requires minimum changes to run using Hercules. Servers can be deployed without requiring any special privileges

Finally, performance and flexibility at server-side are targeted by exploiting the parallel I/O capabilities of Memcached servers. Flexibility is achieved by Hercules due to its easiness to be deployed dynamically on as many nodes as

[1] http://azure.microsoft.com



(a) Scenario 1: Azure storage. (b) Scenario 2: Hercules. (c) Scenario 3: Azure storage and Hercules .

Figure 1: Integration scenarios between DMCF and Hercules.

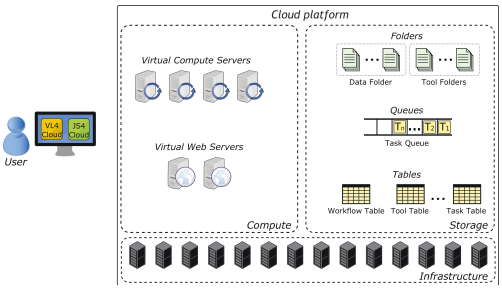


Figure 2: DMCF architecture.

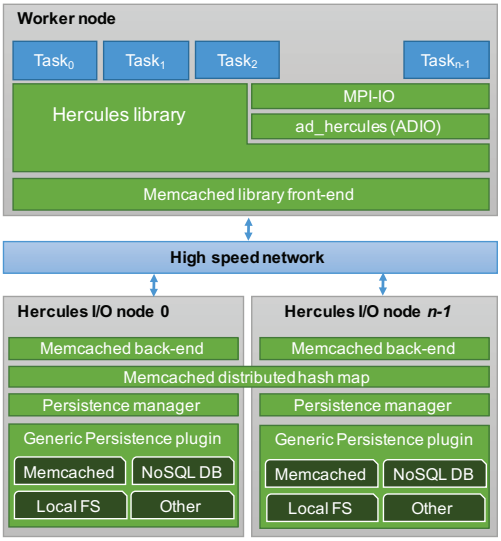


Figure 3: Hercules architecture.

necessary. Each node can be accessed independently, multiplying the total throughput peak performance.

IV. Integration between DMCF and Hercules

The integration between DMCF and Hercules is an ongoing work where different scenarios have been studied in previous works. As can be seen in Figure 1, Hercules and DMCF can be configured according with different deployment scenarios to achieve different levels of integration.

Figure 1(a) shows the original approach of DMCF, where every I/O operation is performed against the cloud storage service offered by the cloud provider (Azure Storage). There are, at least, four disadvantages about this approach: proprietary interfaces, I/O contention in the service, lack of configuration options, and persistence-related costs unnecessary for temporary data.

Figure 1(b) shows a second scenario with the use of Hercules as the default storage for temporary generated data [4]. Hercules I/O nodes can be deployed on as many VM instances as needed by the user depending on the required performance and the characteristics of data.

Figure 1(c) shows a third scenario with a tighter integration of DMCF and Hercules infrastructures. In this scenario, initial input and final output are stored on persistent Azure storage, while intermediate data are stored on Hercules in-memory nodes. Hercules I/O nodes share virtual instances with the DMCF workers.

Previous work [5] explored the third scenario outlined above. However, in order to simplify the implementation of the solution, some workarounds were explored: each time

that one worker needed to access data (read/write operations over a file), it copied the whole file from Hercules servers to the worker local storage. This approach may greatly penalize the potential performance gain in I/O operations for two main reasons:

- *Data placement strategy*. The original Hercules data placement policy distributes every partition of a specific file among all the available servers. This strategy has two main benefits: avoids hot spots and improve parallel accesses. In an improved DMCF-Hercules integration, whole files can be stored on the same Hercules server.
- *Data locality agnosticism*. Data-locality will not be fully exploited until the DMCF scheduler is tweaked for running tasks on the node that contains the necessary data and/or the data is placed where the computation will be realized.

IV.1 Improved integration strategy

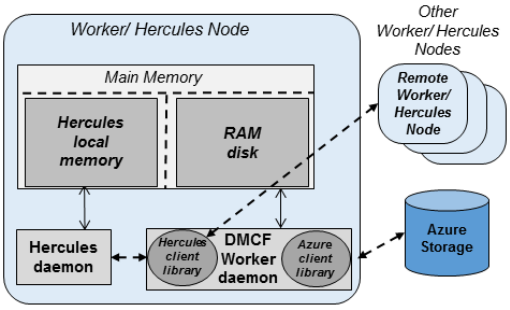


Figure 4: DMCF and Hercules daemons.

Figure 4 describes the proposed improvement to the third scenario of integration between DMCF and Hercules. Four main components are present: DMCF Worker daemon, Hercules daemon, Hercules client library, and Azure client library. The DMCF workers are in charge of executing the tasks of the workflow (data analysis tools/applications), Hercules daemons act as I/O nodes (storing data in-memory and managing data accesses), the Hercules client library is intended to be used by the applications to access to the data stored in Hercules (query Hercules daemons), and the Azure client library is used to read/write data from/to the Azure storage.

To exploit the potential of the data-aware DMCF-Hercules integration, we propose the use of a RAM disk as generic storage buffer for I/O operations performed by workflow tasks. The objective of this approach is the support of DMCF to any existing tool, supporting even binaries independently of the language used for their implementation, while offering in-memory performance for local accesses.

The logic used for managing this RAM disk buffer is based on the full information about the workflow possessed by the DMCF workers. When every dependency of an specific task is fulfilled (every input file is ready to be accessed) the DMCF worker brings the necessary data to the node from the storage (Azure Storage in the first scenario or Hercules in the second scenario). Instead of storing the data in the default file system, as in previous works, we propose storing the data in a RAM disk.

IV.2 Resource optimization challenge and possible solutions

If this solution shows potential performance gains compared with the existing solution, the need of duplicated memory regions (RAM disk and Hercules local memory) can be avoided. We propose three different approaches for solving this challenge:

- Modify Hercules daemons to use the RAM disk memory region as default storage, avoiding the necessity of the *Hercules local memory* region. Hercules daemon can store data in the RAM disk instead of using the Hercules local memory. Any data stored in the RAM disk can be transparently accessed by workflow tasks.
- Modify the code of the workflow tasks (tools/applications) to use the Hercules client library for performing every data access directly over Hercules I/O nodes, avoiding the use of a RAM disk. The main disadvantage is the limited support of existing applications, requiring the modification and re-compilation of every application/tool executed as workflow task.
- Offer the memory managed by Hercules as a storage device, accessed transparently by the workflow tasks. If the Hercules memory subsystem can be mounted as a storage device in every DMCF worker, the applications/tools can access data stored in Hercules in the same way as data stored in any other file system. This approach can be implemented as a FUSE interface or as on-the-fly patching of POSIX I/O operations.

The implementation and evaluation of this approaches is out of the scope of this work, but will be studied in the future.

IV.3 DMCF execution mechanisms and data-aware scheduling

We propose novel workflow-aware task and data placement mechanisms that combine DMCF load-balancing capabilities and Hercules data and metadata distribution functionality for implementing various locality-aware and load-balancing policies. Data placement mechanisms focus in grouping data related to the same task, while the locality-aware scheduler policy targets the co-location of compute task in the nodes where the data can be found in-memory.

In the new execution mechanism proposed the DMCF Worker cyclically checks whether there are tasks ready to be executed in the Task Queue. If so, a task is removed from the Task Queue and its status is changed to 'running'. To take advantage of data locality, the task removed from the queue is the one having the highest number of inputs locally. This differs from the original data-locality agnostic scheduling policy adopted in DMCF, as described in [6], in which each Worker picks and executes the task from the queue following a FIFO policy.

Then, the transfer of all the needed input resources (files, executables and libraries) is performed from their location (Hercules local or remote node) to two local folders and the Worker locally executes the task and waits for its completion.

V. Evaluating the integration between DMCF and Hercules

In this section we show the evaluation results of the integration between DMCF and Hercules. For this evaluation, we have emulated the execution of a data analysis workflow using three alternatives:

- Azure-only scenario: every I/O operation of the workflow is performed by DMCF using the Azure storage service.
- Locality-agnostic Hercules scenario: a full integration between DMCF and Hercules is exploited, where each intermediate data is stored in Hercules, while initial input and final output are stored on Azure. DMCF workers and Hercules I/O nodes share resources (they are deployed in the same VM instance), however, every I/O operations is performed over remote Hercules I/O nodes through the network.
- Locality-aware Hercules scenario: based on the same deployment as the previous case, this scenario simulates a full knowledge of the data location, and executes every task in the same node as the data are stored, leading to fully local accesses over temporary data. Based on this locality exploitation, every I/O operation is performed in-memory instead of through the network.

The goal of this evaluation is to better understand the potential performance improvements in different scenarios where the Hercules I/O accelerator is combined with the DMCF scheduler.

The evaluation is based on a data mining workflow that analyzes n partitions of the training set using k classification algorithms so as to generate kn classification models. The kn models generated are then evaluated against a test set by a model selector to identify the best model. Then, n predictors use the best model to produce in parallel n classified datasets. The k classification algorithms used in the workflow are C4.5 [7], Support Vector Machine (SVM) [8] and Naive Bayes [9], that are three of the main classification algorithms [10]. The training set, test set and unlabeled dataset, which represent the input of the workflow, have been generated from the *KDD Cup 1999*'s dataset[2], which contains a wide variety of simulated intrusion records in a military network environment.

The workflow is composed of $3 + kn + 2m$ tasks. In the specific example, where $n = 20$, $k = 3$, $m = 80$, the number of generated tasks is equal to 223.

Figure 5 shows the VL4Cloud version of the data mining workflow. The visual formalism clearly highlight the level of parallelism of the workflow, expressed by the number of parallel paths and the cardinality of tool array nodes.

Once the workflow is submitted to DMCF using either JS4Cloud or VL4Cloud, DMCF generates a JSON descriptor of the workflow, specifying which are the tasks to be executed and the dependency relationships among them. Thus, DMCF creates a set of tasks that will be executed by workers.

Table 1 lists all the read/write operations performed during the execution of the workflow on each data array. Each row of the table describes: i) the number of files included in the data array node; ii) the total size of the data array; iii) the total number of read operations performed on the files included in the data array; and iv) the total number of write operations performed on the files included in the data array. As can be noted, all the inputs of the workflow (i.e., *Train*, *Test*, *UnLab*) are never written on persistent storage, and the output of the workflow (i.e., *ClassDataset*) is never read.

The simulation results are based on synthetic bandwidth measurements performed over the Azure infrastructure. The benchmark application performs write and read operations over a 256 MB file with a 4 MB chunk size. We have deployed the application on Azure D2_v2 VM instances. The results can be found in Table 3 and represent the expected I/O

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99

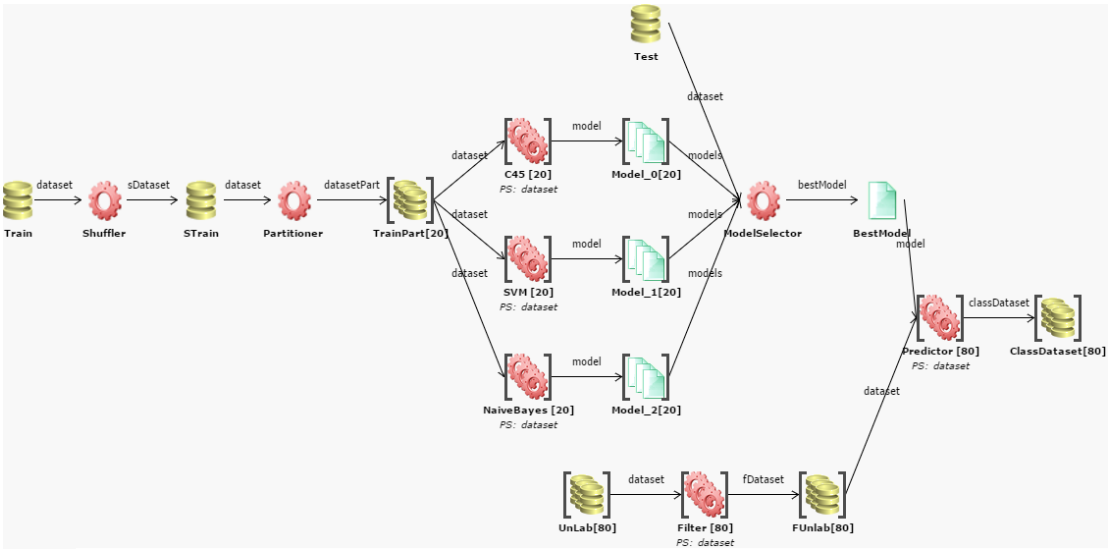


Figure 5: Classification VL4Cloud workflow.

Table 1: Read/write operations performed during the execution of the workflow.

Data node	N. of files	Total size	Number of read operations	Number of write operations
Train	1	100MB	1	-
Strain	1	100MB	1	1
TrainPart	20	100MB	60	20
Model	60	≈20MB	60	60
Test	1	50MB	1	-
BestModel	1	300KB	80	1
UnLab	80	8GB	80	-
FUnLab	80	≈8GB	80	80
ClassDataset	80	≈6GB	-	80

Table 2: Read/write operations performed during the execution of the workflow.

Task Node	N. of istances	Execution times in secs
Shuffler	1	1
Partitioner	1	1
C45	20	288
SVM	20	600
NaiveBayes	20	791
Filter	80	104
ModelSelector	1	9
Predictor	80	2,321

performance of the application when deployed over each evaluated scenario.

Table 3: Synthetic bandwidth measurements performed over the Azure IaaS platform.

Solution	Write op.	Read op.
Azure storage	30 MB/s	60 MB/s
Hercules remote	180 MB/s	175 MB/s
Hercules local	1,000 MB/s	800 MB/s

Figures 6, 7, and 8 show different details of the same experiment where the previously introduced workflow is simulated in different infrastructures. The configurations of the infrastructures range from 1 to 32 DMCF workers. Every DMCF is deployed over a different VM instance, and one Hercules I/O node is deployed on each VM, sharing resources with the DMCF worker. Three different scenarios are studied, as previously presented: Azure-only (labeled as *Azure*), Locality-agnostic Hercules (labeled as *Hercules remote*), and Locality-aware Hercules (labeled as *Hercules local*).

Figure 6 presents an estimation of the total execution time scaling the available resources. The figure shows how the differences in total execution time estimated for every case are narrow, but the cases where the Hercules I/O accelerator is applied are always in front of the Azure-only solution, resulting in up to 8% improvements in total execution times for the best scenarios where the data locality is fully-exploited

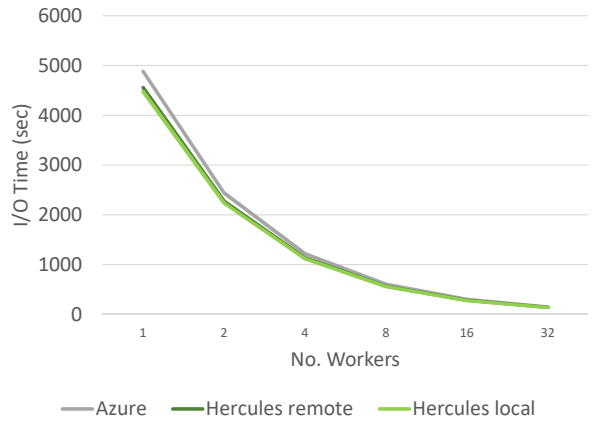


Figure 6: Estimated execution time of the workflow deployed over different scenarios, using up to 32 DMCF workers.

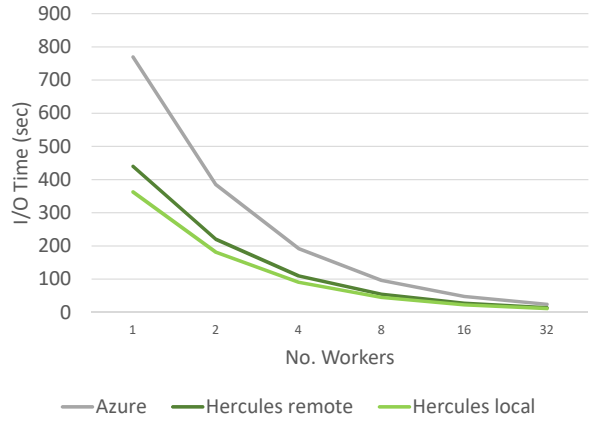


Figure 7: Estimated time required by the application to perform every I/O operation. Different scenarios have been evaluated, using up to 32 DMCF workers.

for temporary files, while 6% reductions in execution time are obtained in locality-agnostic scenarios.

Figure 7 presents an estimation of the time required by the application to perform every I/O operation of the application, and Figure 8 increases the level of detail, showing only the operations affected by the deployment of the Hercules I/O accelerator: I/O operations performed strictly over temporary files. The deployment of Hercules is translated in up to 52% reductions in the time spent in I/O operations when fully exploiting data locality (95% over temporary files) and up to 42% in locality-agnostic scenarios (77% over temporary files).

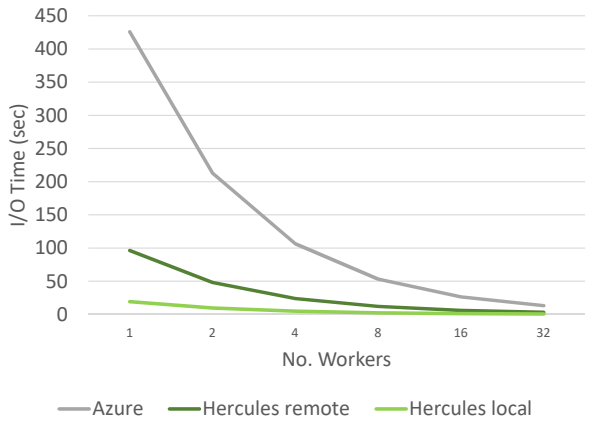


Figure 8: Estimated time required by the application to perform I/O operations strictly over temporary files (I/O operations affected by Hercules). Different scenarios have been evaluated, using up to 32 DMCF workers.

In order to better show the impact of the Hercules I/O accelerator, Figure 9 presents a breakdown of the total execution time, detailing the time spent on each of the tasks executed by the workflow application: computation tasks, I/O tasks over input/results files stored in Azure Storage, and I/O operations performed over temporary files, stored in Hercules when available. This figure clearly shows how the time required for I/O operations over temporary files, the only operations affected by the Hercules accelerator, are reduced to be almost negligible during the execution of the workflow, showing a great potential for increasing the I/O performance in data-intensive applications with large amounts of temporary data. Should be noted how the axis in Figure 9 starts in the second 230, in order to zoom in the top part of the figure, where the I/O times are depicted. The time excluded from the figure (seconds 0 to 230) are spent on CPU operations.

Based on these estimations, we can conclude that the deployment of the Hercules I/O accelerator can greatly benefit the execution of data-intensive applications when a large amount of temporary data is present. The evaluation also shows that, with proper locality-aware mechanisms, the I/O performance can be further improved, exploiting data locality through in-memory computation.

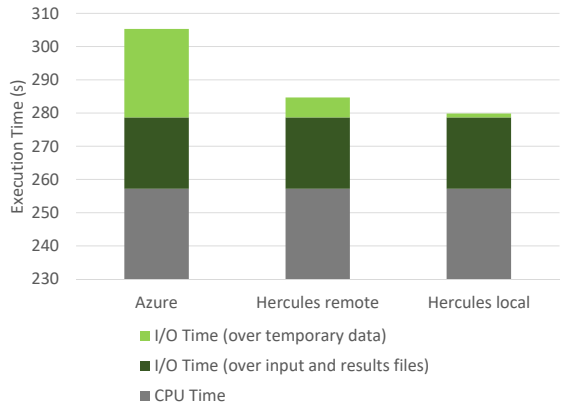


Figure 9: Breakdown of the estimated total execution time deployed over 16 worker nodes sharing resources with 16 Hercules I/O nodes. The breakdown shows the time required by the application to perform compute tasks, I/O tasks over input/result files, and I/O tasks over temporary files (affected by Hercules).

VI. Conclusions

This work presents an evolution of the integration of the Data Mining Cloud Framework (DMCF) with the Hercules in-memory I/O accelerator. The DMCF task scheduler has been improved in combination with Hercules modifications in order to expose and exploit data locality for data-intensive applications.

The evaluation shows an estimation of the potential improvements in I/O performance when data locality is fully exploited during the execution of a data-intensive workflow application deployed over the DMCF-Hercules solution.

Future work should focus on the execution of a real data-intensive application and the evaluation of the improvements achieved by the proposed locality-aware mechanisms. Additionally, an evaluation of the cost of deploying Hercules should be performed, in contrast with the Azure storage-only approach.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Js4cloud: Script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency and Computation: Practice and Experience*, 27(17):5214–5237, 2015.

[2] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. A hierarchical parallel storage system based on distributed memory for large scale systems. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 139–140, New York, NY, USA, 2013. ACM.

[3] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[4] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Jesus Carretero, Domenico Talia, and Paolo Trunfio. Evaluating data caching techniques in dmcf workflows using hercules. In *In Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)*, pages 95–106, Krakow, Poland, 2015.

[5] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Domenico Talia, and Paolo Trunfio. Exploiting in-memory storage for improving workflow executions in cloud platforms. *The Journal of Supercomputing*, pages 1–20, 2016.

[6] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. A workflow management system for scalable data mining on clouds. *IEEE Transactions On Services Computing (IEEE TSC)*, 2016.

[7] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[8] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[9] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[10] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007.